Part 1: Theoretical Understanding

1. Short Answer Questions

   Q1: Explain the primary differences between TensorFlow and PyTorch. When would you choose one over the other?

   • Differences:

   1. Programming Style: While TensorFlow has historically used static graphs (though TensorFlow 2.x supports eager execution), PyTorch uses dynamic computation graphs (eager execution).
   2. Debugging: In general, PyTorch is simpler to debug with Python tools. Debugging TensorFlow's static graph approach necessitates additional setup.
   3. Syntax and Readability: PyTorch is thought to be more beginner-friendly and Pythonic.

   • When to Choose:

   1. Select PyTorch for academic research, quick prototyping, and situations where usability is crucial.
   2. Select TensorFlow if you want scalability, production deployment, and integration with tools such as TensorFlow Serving, TensorFlow Lite, and TensorBoard.

   Q2: Describe two use cases for Jupyter Notebooks in AI development.

   1. Exploratory Data Analysis (EDA): AI developers can analyze datasets, plot graphs, and clean data in real time with Jupyter's interactive data visualization capabilities.

2. Model Prototyping and Experimentation: While using markdown to document the process, developers can create and test brief code fragments for hyperparameter tuning and machine learning model training.

Q3: How does spaCy enhance NLP tasks compared to basic Python string operations?

Beyond simple string operations like split() and find(), spaCy offers linguistically-informed, high-level abstractions like tokenization, named entity recognition (NER), part-of-speech tagging, and dependency parsing. It handles linguistic peculiarities that simple string methods are unable to handle and employs pretrained models for precise and effective natural language processing.

2. Comparative Analysis

- Target Applications

Feature	**Scikit-learn**	**TensorFlow**
Focus Area	Classical machine learning	Deep learning and large-scale neural networks
Typical Models	Decision trees, SVMs, logistic regression, k-NN	Neural networks (CNNs, RNNs, Transformers, etc.)
Best For	Structured/tabular data	Image, text, speech, and sequence data

Feature	Scikit-learn	TensorFlow
Preprocessing	Strong tools for data cleaning, feature selection	Requires external tools (like tf.data) or manual work

- Ease of Use for Beginners

Feature	Scikit-learn	TensorFlow
API Design	Simple, consistent, Pythonic	More complex; lower-level APIs can be intimidating
Model Training	One-liner .fit() syntax	More setup required (model.compile(), fit(), callbacks)
Learning Curve	Gentle; great for learning the fundamentals	Steeper; better for those with some ML background
Out-of-the-box Use	Many models require minimal configuration	Models need careful architecture design

- Community Support

Feature	Scikit-learn	TensorFlow
Maturity	Older (since ~2007), very stable	Modern (since 2015), continuously evolving

Feature	Scikit-learn	TensorFlow
Docs & Tutorials	Clear and concise documentation	Extensive official docs + TensorFlow Hub and Model Garden
Ecosystem	Integrates well with NumPy, Pandas, Matplotlib	Huge ecosystem: Keras, TF Lite, TF Serving, TFX, etc.
Community Size	Large, strong in academia	Massive, with contributions from Google and industry partners

Ethical Considerations

Identify potential biases in your MNIST or Amazon Reviews model. How could tools like TensorFlow Fairness Indicators or spaCy’s rule-based systems mitigate these biases?

1. MNIST (Image Classification)

   Despite MNIST's relative cleanliness, biases may still exist:

   Unbalanced dataset: A model that performs better on certain digits may result from overrepresentation of those digits.

   Style bias: MNIST does not have demographic data to monitor handwriting styles, which can vary by age, gender, or culture (e.g., different ways of writing the digit "1" or "7").

Model bias: The model may fail to generalize to handwriting styles from underrepresented groups because it overfits to particular pixel patterns.

2. Amazon Reviews (Sentiment Analysis)
   This NLP task is more prone to bias:

   Demographic bias: Reviews written in slang or dialects specific to a community (such as African American Vernacular English) may perform worse in the sentiment model.

   Racial or gender bias: Due to biased training data, some terms or expressions that are connected to particular groups may be mistakenly classified as more positive or negative.

   Labeling bias: Subjective interpretations may be introduced into sentiment labeling by human annotators.

Mitigation Strategies

- **TensorFlow Fairness Indicators**

  It aids in assessing model fairness across data slices and is most appropriate for structured prediction tasks (such as sentiment analysis):

  Amazon Reviews use case:

  When dialectal or demographic characteristics (such as the reviewer's gender, location, or writing style) are included as metadata, Fairness Indicators can:

Draw attention to performance disparities (such as a lower F1-score for reviews written by women).

Calculate fairness indicators such as demographic parity or equal opportunity.

This enables you to adjust training data or fine-tune your model as necessary.

Less relevant to MNIST unless expanded with metadata, which is usually absent (e.g., writer's age/gender).

- **spaCy's Rule-Based Systems**

These systems add to or filter machine learning predictions using phrase matchers, token rules, and patterns.

Amazon Reviews use case:

Prior to training, you can identify and flag problematic phrases or biased language.

Preprocessing based on rules can:

Reduce dialect bias by normalizing idioms and slang.

Emotionally charged words that could skew the sentiment score should be eliminated or marked.

MNIST use case: Since spaCy is a text processing tool, it is not relevant.